Improving Hardware Security by Using Hidden Information of Computer System

[1]Ilya Levin, [2]Osnat Keren, [2]Vladimir Sinelnikov
*[1]Tel Aviv University, ilia1@post.tau.ac.il
[2]Bar Ilan University, kereno@eng.biu.ac.il
[2] Bar Ilan University, sinel@zahav.net.il

Abstract

An important direction of improving security of computer systems is detection of Trojan horse hardware. A Trojan horse is a malicious altering of hardware specification or implementation in such a way that its functionality is altered under a set of conditions defined by the attacker. The paper presents a technique for designing secure systems that can detect an active Trojan that distorts or manipulates their proper operation. The technique is based on utilizing specific information about the system's behavior, which is known to the designer of the system and/or is hidden in the functional specification of the system. A case study of the proposed technique conducted on an arithmetic unit of a microprocessor is provided. The study indicated a high level of Trojan detection with a small hardware overhead.

Keywords: *Trojan Hardware, Design for Security, On-line checking, Hidden Information*

1. Introduction

The modern society is characterized by the most intensive use of information technologies. As a result, data and knowledge has higher value than material products, and informational infrastructures forming base to various services become the main target of cyber attackers (cyber space terrorists, criminals, espionage, state supported cyber attackers, etc.) For example, disabling or destruction of systems of communication, energy and water supply, or damaging databases of systems of medical institutions very often turns to be as critical as physical destruction of them

Both the cyber attacks and cyber defense of infrastructure of the computerized services are twin-tasks of Information Technologies. The goal of cyber attackers is to destroy or damage information infrastructures. Today, when terrorism is becoming more and more sophisticated, when threats to security can be planned in advance and supported by significant monetary resources, it should be noted that informational infrastructures may be damaged not only "from outside", but also "from inside", by introducing erroneous or even some “controlled misbehavior” into their functions (so-called "Trojan hardware"). For example, instead of investing great efforts into breaking encryption algorithms which are used by a target device, intruders/terrorists may easily note, that the device hardware that performs the encryption, remains vulnerable to malicious modification of its circuitry (almost at any stage of its design, production, or usage) and to side-channel cryptographic attacks, as well as to modification of data stored in the memories of the hardware (HW). The hardware therefore becomes exposed to attacks on it.

The problem of protecting the hardware from malicious attacks becomes even more critical if the hardware is reconfigurable. In the reconfigurable systems, complex software and hardware are coupled and implemented on a single microchip which can be shared by both software and hardware engineers. Such systems can be accessed through the Internet and can be modified from outside. This makes them even more vulnerable because of the static and dynamic changes that can be introduced at any stage of the design. For example, as it was announced by leading companies, extensible processing platforms (e.g. Zynq-7000 of Xilinx, ARM A9 SoC FPGA of Altera) incorporate dual core ARM A9 Cortex and powerful field-programmable gate arrays (FPGA) achieving complexity of 6.8 billion transistors on a single chip. Software and hardware can be developed independently of each other and in parallel. Since nowadays the design starts of FPGA-based systems exceed the similar characteristics for ASICs and ASSPs-based systems by an order of magnitude and in accordance with forecasts this tendency will continue in future, we can expect eventually an unpredictable and uncontrolled use of very powerful hardware/software resources capable to interference with other systems through global networks.

Consequently, uncontrollable number malicious insider attackers will be able to access such systems with severe consequences to the systems. Thus, hardware has to be protected not only from outside but also from inside even within the same microchip.

Trojan horse hardware is a malicious modification of the circuitry of an integrated circuit. The Trojan hardware is created by an adversary that adds or deletes some transistors or gates to the original design and thus changes its functionality. The adversary may also change parameters of the original circuitry in a way that harms the reliability of a system. When the Trojan is active, the output of the system is erroneous. In this sense, an active Trojan can be modeled as arbitrary error vector that is added on top the correct *output* vector and distorts it. Since the system functions correctly when the Trojan is not activated, it is difficult to detect a Trojan by off-line testing. This motivates researchers to classify, model, and analyze Trojans and their effect on the behavior of the system, as well as to develop methods to detect a Trojan HW [4][5][6][7][9][15][16][17]. Trojan HWs can be classified into three major types: a) HW broadcasting to the attacker some internal signals, b) HW distorting the functionality of the system, and c) HW which destroying the chip. In this paper we deal with Trojans that distort the operation of the system (type b).

Various Trojan detection methods can be applied at different levels of chip design and fabrication. At the fabrication level a Trojan may be detected by performing parametric analysis at fabrication stage, this however is extremely time consuming and expensive. Another approach is based on side channel analysis; this method is effective for detecting functional Trojans only in small circuits since it requires isolating the tested area from other components. Automatic Test Pattern Generation (ATPG) is another method that can be used for Trojan detection. This method is efficient only if it activates the Trojan, otherwise the system functions correctly and the Trojan is undetected. In this paper, we present a different approach. Our approach is based on on-line testing. We suggest utilizing a Concurrent Error Detection (CED) [2][14] mechanism, which is normally used to cope with faults (permanent or transient) in the circuitry to detect active Trojans.

In general, there are three approaches to the design of CED schemes: the first is based on analyzing all the possible error events in a given circuit, the second is based on analyzing the possible combinations that may appear on the output lines of each logic block, and the third approach is based on analyzing the functionality of the whole system. Clearly, the first two methods are more suitable for trapping errors that are caused by faults, since a) they heavily relay on the actual implementation of the combinatorial circuit and b) they check the correctness of the current output. The third approach, which considers the functionality of the system, can detect an *incorrect sequence of outputs*, even if each output by itself is legal. The CED mechanism presented in this paper belongs to the third class of CEDs.

In [13], the idea of utilizing hidden information about the system functioning for detecting Trojan HW was proposed. Such implicit information about the correct functionality of the system is not necessary included in an initial specification of the system. This information is known to the designer, and in some cases can be retrieved from the functional specification of the system. However, this information disappears or become inaccessible when the system is represented in HDL and synthesized by conventional CAD tools.

In the present paper, we develop the above approach. We show how the hidden information can be used to provide an additional layer of security. We present a case study – a CPU and show that one can increase the level of security by adding a negligible amount of logic.

The paper is organized as follows. The next section formulates the problem and reviews related work. Section 3 describes the suggested design for security methodology. Section 4 presents a motivation example and Section 6 concludes the paper.

2. Problem Formulation and Related Work

In general, any system has components that can be classified either as parts of the datapath (DP) or as control unites (CUs). A Trojan can distort or manipulate the correct operation of the system by distorting the inputs and/or the outputs of the control unit as well as its internal states. In other words, the operation of the system in the presence of an active Trojan can be modeled as unknown logic blocks that are placed in between the CU and DP. Figure 1 shows the conventional partition of the system into DP and CU. The control unit that is a finite state machine (FSM) consists of two blocks, combinatorial logic and memory elements that store the state of the system. In the figure, *PS* represents the present state and *NS* represents the next state. The inputs to the control unit are external (*Xe*) signals

and internal signals (*Xi*) that are generated by the datapath. The control unit generates microinstructions, denoted by *Yi*, which control the datapath operation.

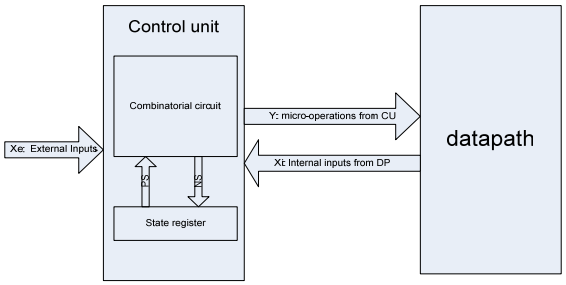


Figure 1. Original system

An active Trojan can interfere or manipulate the correct operation of the system in two ways. It may *directly* distort the outputs of the control unit (that is, modify the microinstructions or the next state variables), or/and it may cause the control unit to generate different outputs by *changing the inputs* of the control unit.

We assume that the Trojan HW designer is not familiar with the specification of the system. That is, the Trojan HW does not generate (for at least a period of time) a legal sequence of input/outputs vectors. Therefore, its effect can be modeled as an arbitrary error that is added to the correct signals. This assumption is essential since if the Trojan generates only legal sequences inputs or outputs it cannot be detected at all. This allows modeling the effect of the active Trojan on the inputs and outputs as appearance an additive error in the correct information (see Figure 2).

We assume that there is no mandatory requirement to detect the Trojan immediately at the first cycle it was activated. Therefore, the goal is to design a low-cost CED mechanism that is capable of detecting a Trojan with high probability over a predefined time interval. This goal can be achieved by protecting only a sub-set of the possible legal input and output words.

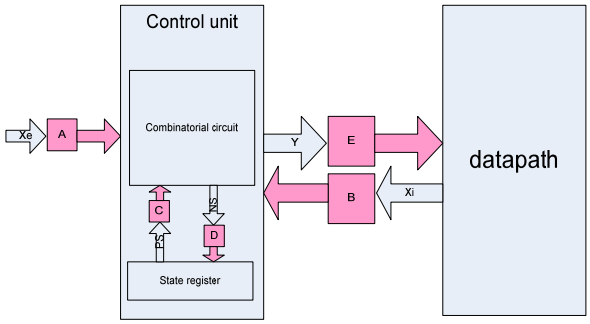


Figure 2. System with Trojan blocks. The red blocks show Trojan effect

We distinguish between two cases: a) the ability of a block to verify that it receives a correct (external) input or a correct sequence of inputs, and b) the ability of a block to verify its own correct operation.

The problem of detecting an erroneous input (case a) requires a redundant representation (coding) of the information that the input carries. The encoding is done by the block that generates the information and the decoding, i.e. the check of correctness, is done by the receiving block. Concurrent Error Detecting (CED) codes are usually systematic codes that add a number of redundant bits to the original information bits. The codes are designed to cope with a specific error model. Extensive study has been done on codes that can detect errors that are caused faults or transient errors in the circuit **[1]**[11][12]**[**18**]**. In [10], a family of *robust codes* and partially robust codes that can detect arbitrary errors from non-stationary source with a minimal undetected error probability has been introduced. In this paper we deal with the second problem – the ability of a block to *detect its incorrect operation*. Namely, on top of the layer that detects erroneous external inputs that enter the control unit or the datapath, we add another layer of security that detects an erroneous operation of the block itself. While

the robust codes (e.g. [10]) can detect an active Trojan modeled by blocks A and E in Figure 2, our mechanism is designed to detect a Trojan whose effect is modeled by blocks B-D in the figure. To illustrate the necessity of such mechanism, consider a Trojan whose effect can be modeled by block C. Such a Trojan may change the state of the system. This however will not produce an illegal microinstructions and it will not move the system to a wrong state. In other words, such a Trojan cannot be detected outside the control unit by using a conventional CED mechanism. Indeed the only way to detect it is by using information that is available in the functional description of the system or is known to the developer of the system.

3. Design for Security Methodology

To introduce our approach, we use the representation of the given CU in a form of Algorithmic State Machine (ASM) chart. Using ASMs for specification was not so common in last decades. This can be explained by two main reasons: a) wide use of high-level description languages; b) domination of the *architectural-approach* in chip design, in which a system is described in a form of interacting elements and connections between them. An alternative approach to the architectural-approach is the *algorithmic-approach*. This uses a set of ASMs, each represents a different algorithm that is performed by the system. For example, designing a CPU can be done by dividing the system for two interacting subsystems CU and Operational Unit (OU), i.e. a DP, and constructing an algorithmic description of this interaction for each instruction (or subset of instructions) supported by this CPU.

An Algorithmic State Machine (ASM) chart is a directed connected graph containing an initial vertex (initial operator Y_b), a final vertex (end operator Y_e), a finite set of operator vertices and conditional vertices. One of the input variables is written in each conditional vertex. Each operator (microinstruction) $Y_i$corresponds to an output vector. The operator is defined a subsets of the set of micro-operations $Y = \{y_1, \ldots, y_N\}$. Any ASM can be transformed to the FSM form [3]. The FSM specification is the list of transactions, which are paths of the initial ASM. An example of an ASM is given in Figure 3. This ASM describes a microcontroller. There are 5 input variables, $X = \{x_1, \ldots, x_5\}$, 6 microinstructions $Y_1, \ldots, Y_6$ and 8 micro-operations $Y = \{y_1, \ldots, y_8\}$. This ASM can be transformed into an FSM with 6 states.

Each ASM operator vertex is performed during a single clock. In general, all input variables of the CU can change their values during a clock. Nevertheless, in real designs, usually it is not true - very often, a large subset of input variables cannot change their values. ASMs do not contain this information explicitly. Nevertheless, this hidden information is known to the developer.

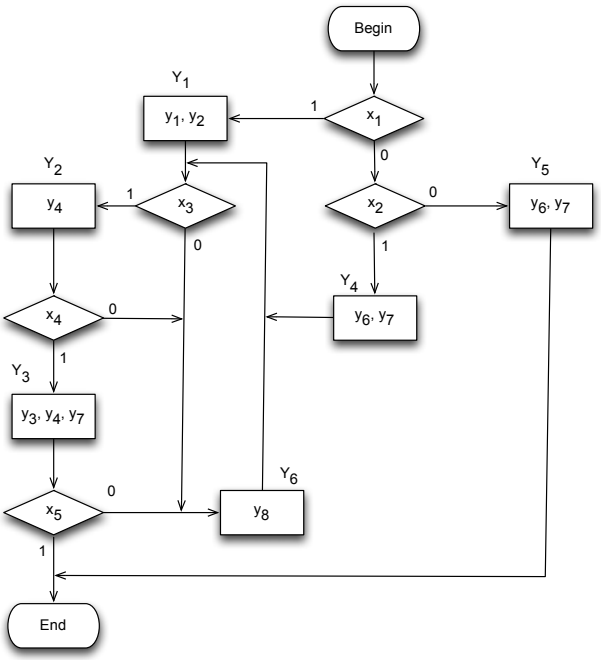


Figure 3. An example of Algorithmic State Machine

The initial point of our study is the idea that this additional information can indicate the presence of Trojan. Such additional information is, actually, a set of variables, which don't change values as a result of performing of certain operators of ASM.

Let the following conditions are met: a) before performing an operator, values of some variables are known in advance, and b) these variables belong to the set of variables that don't change values as a result of performing this operator. Then, checking variables immediately after performing the operator, gives valuable information about correctness of the system's operation.

In this paper, we propose to add a number of conditional vertices into the given ASM; these vertices check values of such input variables that don't change values when the system functions properly.

We describe our idea on the example of ASM shown on Figure 3.

First of all, we construct a matrix of the given ASM [3]. Both columns and rows of the matrix correspond to operators of the ASM. (The rows include the initial operator Y_b, the columns – the end operator Y_e. Each specific cell α_{ij} comprises a transition function, equal to 1 when the operator Y_j is performed immediately after the operator Y_i.

Table 1. Matrix of ASM from Figure 1

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b	x_1			$\bar{x}_1 x_2$	$\bar{x}_1 \bar{x}_2$		
Y_1		x_3				$\bar{x}_3$	
Y_2			x_4			$\bar{x}_4$	
Y_3						$\bar{x}_5$	x_5
Y_4		x_3				$\bar{x}_3$	
Y_5							1
Y_6		$\bar{x}_3$				$\bar{x}_3$	

Let the list of operators Y_i with the corresponding sets of variables that don't change their values when the operator performs, is as follows:

$$Y_1 \Rightarrow \{x_1\}; Y_2 \Rightarrow \{x_4, x_5\}; Y_3 \Rightarrow \{x_4, x_5\}; Y_4 \Rightarrow \{x_1\}; Y_3 \Rightarrow \{x_2\}; Y_6 \Rightarrow \{x_3, x_4\}.$$

Observing the matrix, we arrived to a conclusion that there are operators that can be potentially used for utilizing the above information for a Trojan detection. These operators are: Y_1, Y_3, Y_4, Y_5. The corresponding columns include variables having the constant value. More specifically: transitions to Y_1 are carried out if $x_1 = 1$; to Y_3 if $x_4 = 1$; to Y_4 if $x_1 = 0$, $x_2 = 1$; to Y_5 if $x_1 = 0$, $x_2 = 0$.

Since the operator Y_1 doesn't change the value of x_1, after performing the operator Y_1, x_1 keeps the value 1. To detect a possible Trojan, we insert an additional conditional vertex containing x_1 into the initial ASM immediately after the operator Y_1. That will keep the normal order of the ASM if $x_1 = 1$ and indicate the presence of a Trojan if $x_1 = 0$. Similarly, we insert another three additional conditional vertices: x_4 - immediately after Y_3, x_1 - immediately after Y_4 and x_2 - immediately after Y_5.

It is important to note that despite the fact that two variables have a constant value on each of transitions to Y_4 and Y_5 only one of the variables does not change the value. Thus only one additional conditional vertex is included after each of the operators in the transformed ASM.

The ASM transformed for Trojan detection is shown in Figure 4.

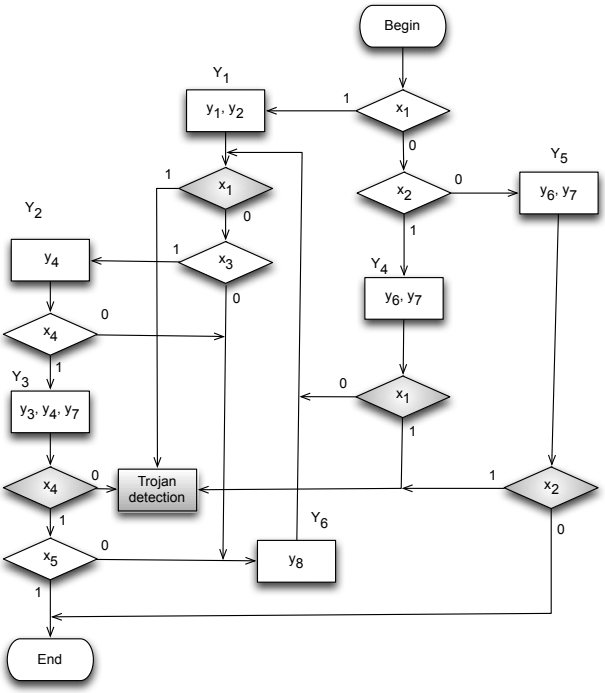


Figure 4. ASM from Figure 3 with Trojan detection property

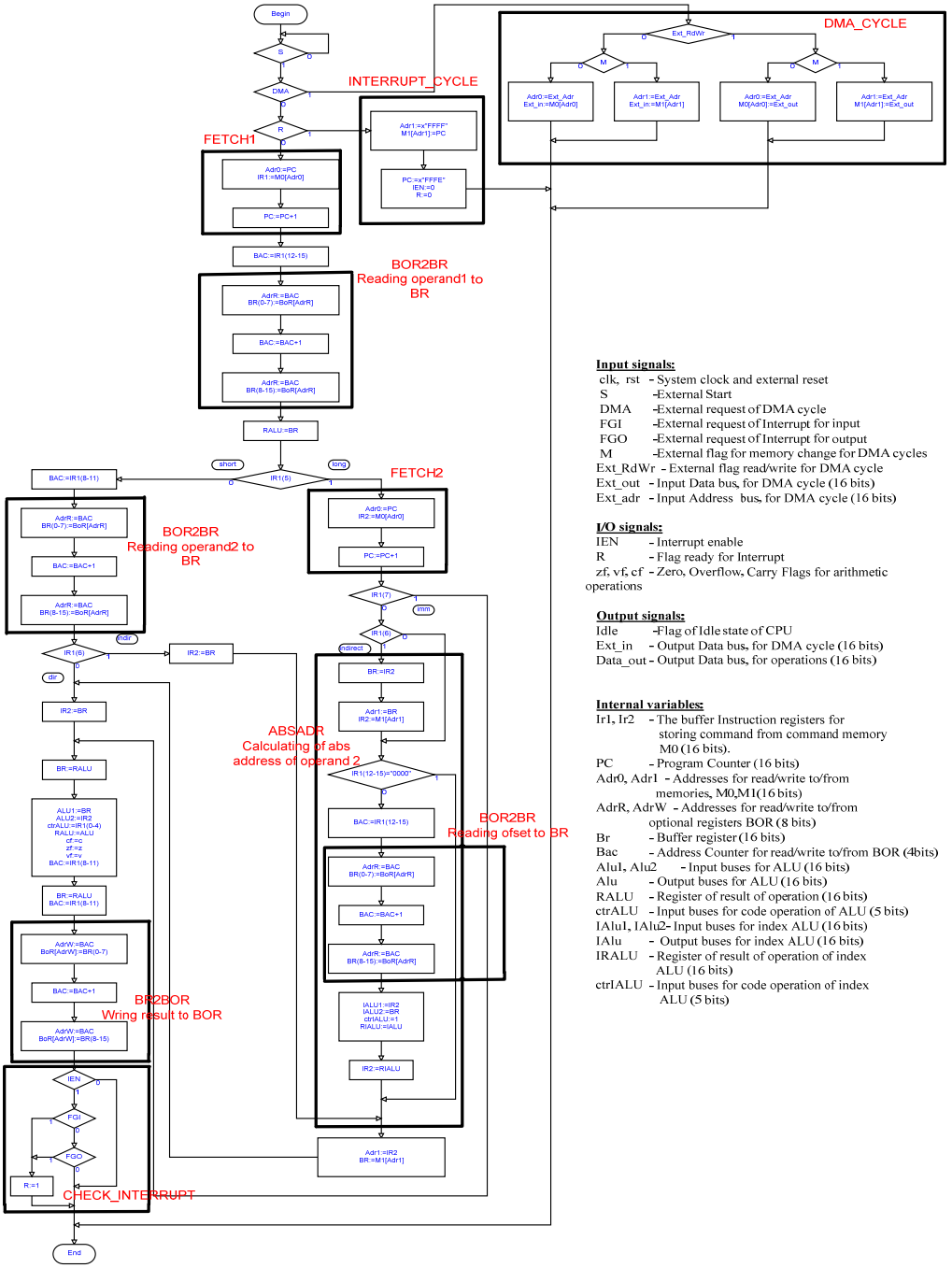
Note that in Figure 4, the additional vertices are highlighted in the gray color.

In the next section, we present the implementation of the above ideas using the example of a certain CPU.

4. Case Study – CPU Design

A functional description of a system by ASM allows representing the system as a set of component ASMs. Each component represents a single aspect of functionality. This allows the Trojan detection mechanism for each component. This simplifies the design since opens a way to handle each of the components separately. We demonstrate the design of the Trojan detection mechanism for a CPU having 16 8-bit registers and two embedded 16-bit memories for data and instructions. The CPU supports short and long formats and the addressing modes: direct, indirect, immediate, direct indexed and indirect indexed.

The ASM shown in Figure 5 is one of ASMs that specify the processor [3].



Input signals:
clk, rst - System clock and external reset
S -External Start
DMA -External request of DMA cycle
FGI -External request of Interrupt for input
FGO -External request of Interrupt for output
M -External flag for memory change for DMA cycles
Ext_RdWr - External flag read/write for DMA cycle
Ext_out - Input Data bus, for DMA cycle (16 bits)
Ext_adr - Input Address bus, for DMA cycle (16 bits)

I/O signals:
IEN - Interrupt enable
R - Flag ready for Interrupt
zf, vf, cf - Zero, Overflow, Carry Flags for arithmetic operations

Output signals:
Idle -Flag of Idle state of CPU
Ext_in - Output Data bus, for DMA cycle (16 bits)
Data_out - Output Data bus, for operations (16 bits)

Internal variables:
Ir1, Ir2 - The buffer Instruction registers for storing command from command memory M0 (16 bits).
PC - Program Counter (16 bits)
Adr0, Adr1 - Addresses for read/write to/from memories, M0,M1(16 bits)
AdrR, AdrW - Addresses for read/write to/from optional registers BOR (8 bits)
Br - Buffer register (16 bits)
Bac - Address Counter for read/write to/from BOR (4bits)
Alu1, Alu2 - Input buses for ALU (16 bits)
Alu - Output buses for ALU (16 bits)
RALU - Register of result of operation (16 bits)
ctrALU - Input buses for code operation of ALU (5 bits)
IAlu1, IAlu2- Input buses for index ALU (16 bits)
IAlu - Output buses for index ALU (16 bits)
IRALU - Register of result of operation of index ALU (16 bits)
ctrIALU - Input buses for code operation of index ALU (5 bits)

Figure 5. ASM of microprocessor

The ASM describes the algorithm for the execution of the *add*, *subtract* and *shift* arithmetic instructions. To simplify the presentation we have circled sets of vertices that correspond to DMA-cycle, Interrupt, Fetch of short and long instructions from the memory, reading and writing data to/from the registers and the algorithm for calculating the address of an operand. The ASM in Figure 5 defines and uses:

- 13 inputs (4 internal and 9 external) inputs out of 18 inputs to the whole CU that supports the complete set of instructions.
- 27 microinstructions, each consists of 45 micro-operations.

- 27 states coded as 1-hot.

Recall that the processor's functioning is described as a system of a number of ASMs each corresponds to a subset of instructions. Thus, when *add/subtract* or *shift* is preformed the execution will be performed according to the algorithm in Figure 5. That is, the values of the variables (input, microinstructions and states) will be determined solely by this ASM. In other words, in a fault free circuit, only an active Trojan may change the expected value of these variables.

In our example, we have added 10 check vertices and two operator vertices to the ASM to verify the correct behavior of the system at each step. These vertices, which form the Trojan detection mechanism, check (on average) 31 out of the 40 inputs (state variables and internal inputs). This results a detection of 85% of possible unwanted changes in the states of the system, and detecting of 22% of possible (unwanted) changes in the inputs to the CU (refer to blocks B-D in Figure 2).

The implementation cost of the Trojan detection mechanism is shown in Table 2 and measured in terms of the number of slices, LUTs and Flip-Flops in a Xilinx Spartan3a, XC3S1400An. In the table, we present the original size of the whole system (DP and CU), the size of the original CU, the CU additional logic to be included to allow CED of the microinstructions sent to the DP. Columns 5 and 6 show the additional logic that supports the check of validity of the inputs (denoted as XCh), and the state variables (denoted as state Ch). The last column shows the cost of the CED checker located right in front of the DP. The additional logic required to protect the states and the inputs of the CU, which may be distorted by a Trojan modeled by blocks B-D in Figure 2 is 24% of the cost of the CU (when measured in slices) and is 0.1% of the cost of the whole system. The additional logic required to protect the whole system against a Trojan modeled by blocks B-E in Figure 2, is 3% of the cost of the whole system.

Table 2. Overhead in CU design

	DP + CU	Orig. CU	CU + coded Y's	X Ch.	state Ch.	Y Ch.
Slices	7293	29	31	2	5	205
FF	8577	49	49	3	3	378
LUTs	5632	33	57	3	10	0

5. Conclusions

This paper presents a novel approach for design for security. Our approach is based on utilizing some information about the system's functioning that is absent in the initial specification. We use an ASM description of a system to present our idea. The additional information actually is a data about input and variables states of the system that keep there values during a normal functioning of the system and are able to change their values only in a case of presence of Trojan hardware in the system.

We developed an approach for inserting some redundancy to the system allowing indicating changes of values of specific known in advance variables. A case study presented in the paper shows a high Trojan coverage with relatively small hardware overhead.

Our future research in the described direction includes development of methods and algorithms for automation of the process the hidden information and methods for synthesis of systems with Trojan detection capability.

6. References

[1] B. Abramov, O. Keren, I. Levin and V. Ostrovsky, "Constructing Self-testing Circuits with the Use of Step-by-step (Cascade) Control", Automation and Remote Control, vol. 70, no. 7, pp. 1217-1227, 2009 .

[2] S. Almukhaizim, P. Drineas and Y. Makris, "Entropy-driven parity-tree selection for low-overhead concurrent error detection in finite state machines", IEEE Transactions on Computer-Aided Design of Integrated Circuits and System, vol. 25, no. 8, pp. 1547- 1554, 2006.

[3] Baranov S. Logic and System Design of Digital Systems, TUT press, 2008.

[4] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M. Hsiao, J. Plusquellic, and M. Tehranipoor, “Protection against hardware Trojan attacks: Towards a comprehensive solution”, IEEE Design and Test, issue 99, 2012.

[5] Bilzor, M., Huffmire, T., Irvine, C., & Levin, T. “Security checkers: Detecting processor malicious inclusions at runtime”. In Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), , pp. 34-39, 2011.

[6] R.S. Chakraborty, S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation", In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design - ICCAD 2009, pp. 113-116, 2009.

[7] R.S. Chakraborty, S. Narasimhan, S. Bhunia, "Hardware Trojan: Threats and emerging solutions”, In Proceeding of the IEEE Int. High Level Design Validation and Test Workshop - HLDVT 2009, pp. 166-171, 2009.

[8] N.K. Jha and S.J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits", IEEE Transaction CAD, vol. 12, no. 6, pp. 878-887, Jun. 1993.

[9] Y. Jin, N. Kupp, Y. Makris, "Experiences in Hardware Trojan design and implementation", In Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust - HOST '09, pp. 50 - 57, 2009.

[10] M.G. Karpovsky, K. Kulikowski, Z, Wang, "Robust Error Detection in Communication and Computation Channels," Keynote paper in the Int. Workshop on Spectral Techniques, 2007.

[11] Kaushik De, Chitra Natarajan, Devi Nair, Prithviraj Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits", IEEE Transaction on Very Large Integration (VLSI) Systems, vol. 2, no. 2, pp. 186-195, June 1994 .

[12] O. Keren, "One-to-many: Context-Oriented Code for Concurrent Error Detection", JETTA - Journal of Electronic Testing. Theory and Applications, vol. 26 no.3, pp. 337-353, 2010.

[13] O. Keren, I. Levin, V. Sinelnikov, “Detection of Trojan HW by using hidden information on the system”, 17th IEEE International On-Line Testing Symposium (IOLTS-2011), Athens, Greece, 192–193, 2011.

[14] P. Lala. Self-checking and Fault-Tolerant Digital Design, Morgan Kaufmann Publishers, 2000.

[15] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhunia, “Improving IC security against Trojan attacks through integration of security monitors”, IEEE Design and Test, vol. 29, issue 5, 2012.

[16] M. Potkonjak, A. Nahapetian, M. Nelson, T. Massey, "Hardware Trojan horse detection using gate-level characterization”, The 46th ACM/IEEE Design Automation Conference - DAC '09, pp. 688-693, 2009 .

[17] M. Tehranipoor, F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection", IEEE Design and Test of Computers, vol. 27, no. 1, pp. 10-25, 2010 .

[18] N.A. Touba, and E.J. McCluskey ”Logic Synthesis of Multilevel Circuits with Concurrent Error Detection”, IEEE Trans. Computer-Aided Design of Integrated Circuits and System, vol. 16, pp. 783-789, 1997.